ROI Models for Neonatal Video Preprocessing

- MediaPipe Face Mesh (BlazeFace + Face Landmarker)
- BlazeFace (lightweight face detector)
- PFLD — Practical Facial Landmark Detector (mobile-optimized)
- Fast-SCNN / MobileNetV3 + DeepLabV3-lite (light segmentation backbones)
- ROI selection methods tuned for rPPG (algorithmic, uses 3D landmarks + head pose)

Why MediaPipe Face Mesh is Most Suitable

High Landmark Density

- Predicts 468 3D landmarks across the face.
- Allows precise ROI selection for cheeks and forehead without pixel-level segmentation.

Mobile-Optimized Architecture

- Designed to run efficiently on mobile GPUs via TFLite GPU delegate.
- Real-time performance: 30–100+ FPS on modern flagship phones.

Low Model Size & Parameters

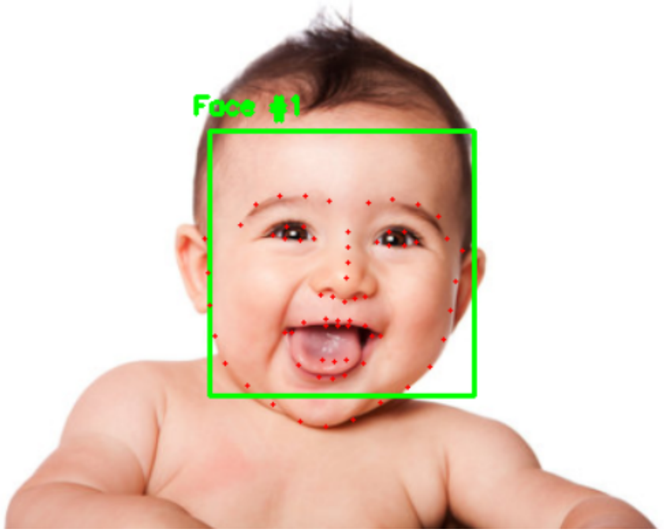
- ~2.3M parameters (~9 MB) — small enough for mobile apps.
- Lightweight compared to larger models like ResNet or heavy segmentation networks.

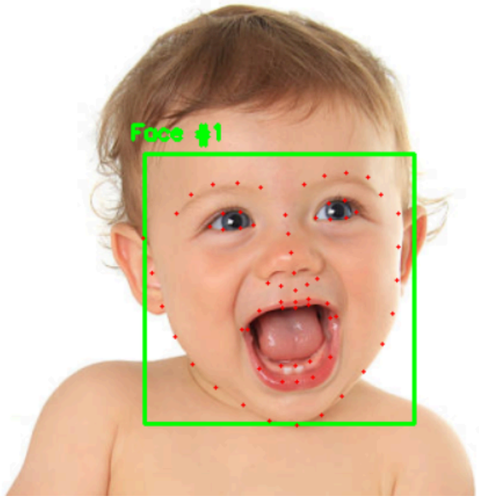
Masking Neonate Eyes to Ensure Privacy

1. Facial Landmark Detection

Use a pre-trained facial landmark detector (e.g., Dlib, OpenCV).

These tools can detect key landmarks around the face, specifically the eyes.

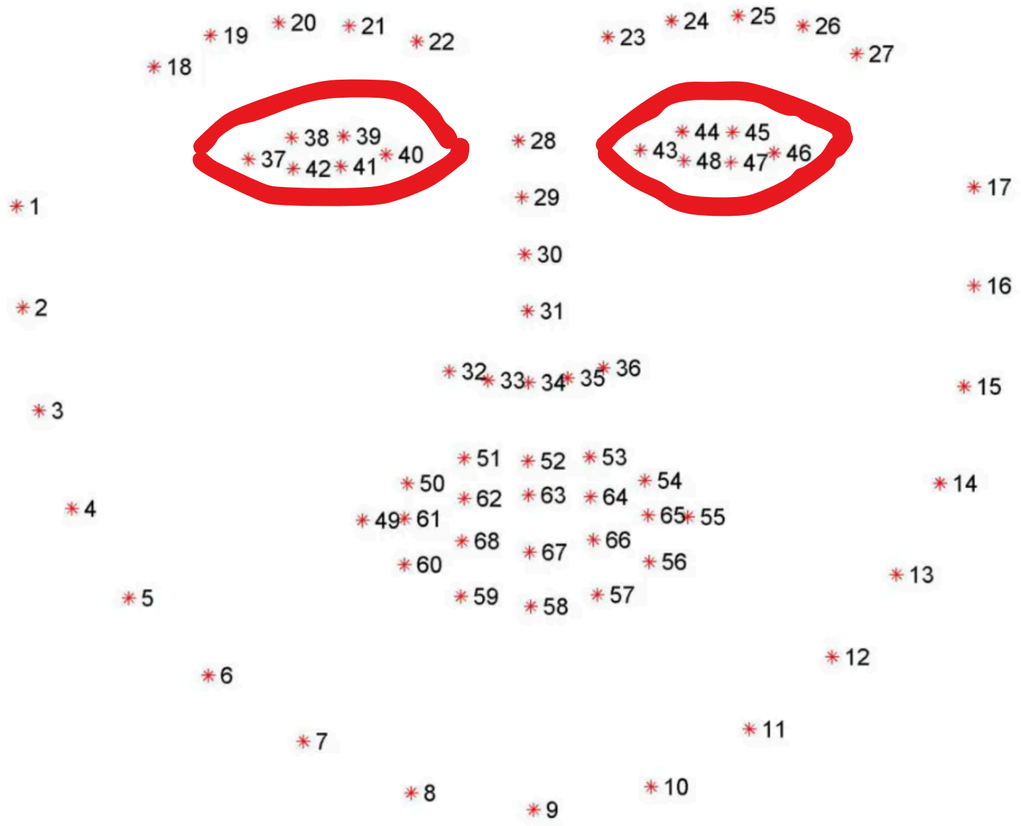




https://github.com/Practical-CV/Facial-Landmarks-Detection-with-DLIB

2.Detect Eyes and Get Coordinates

After detecting the face, extract the coordinates of the eyes using facial landmarks.



- The pre-trained facial landmark detector inside the dlib library estimate the location of 68 (x, y)-coordinates that map to facial structures on the face.

- In Dlib's 68-point model, landmarks 36 to 41 correspond to the left eye, and 42 to 47 correspond to the right eye.

https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/

3.Masking the Eyes

Blurring: Apply a blur effect using OpenCV’s GaussianBlur or boxFilter.

Gaussian Blur:

This technique uses a Gaussian kernel to smooth the image, effectively reducing noise and blurring details. It is applied using cv.GaussianBlur().

Box Filter (Average Blurring):

This method uses a simple box filter to calculate the average of pixel intensities within a defined kernel area, resulting in a uniform blur. It is applied using cv.blur(), requiring the kernel size.

4. Processing the Video

- Extract frames: Read the video frame by frame.
- Apply the mask: For each frame, detect the face and eyes, and apply the mask.
- Reassemble the video: After all frames are processed, stitch them back into a video.

https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html